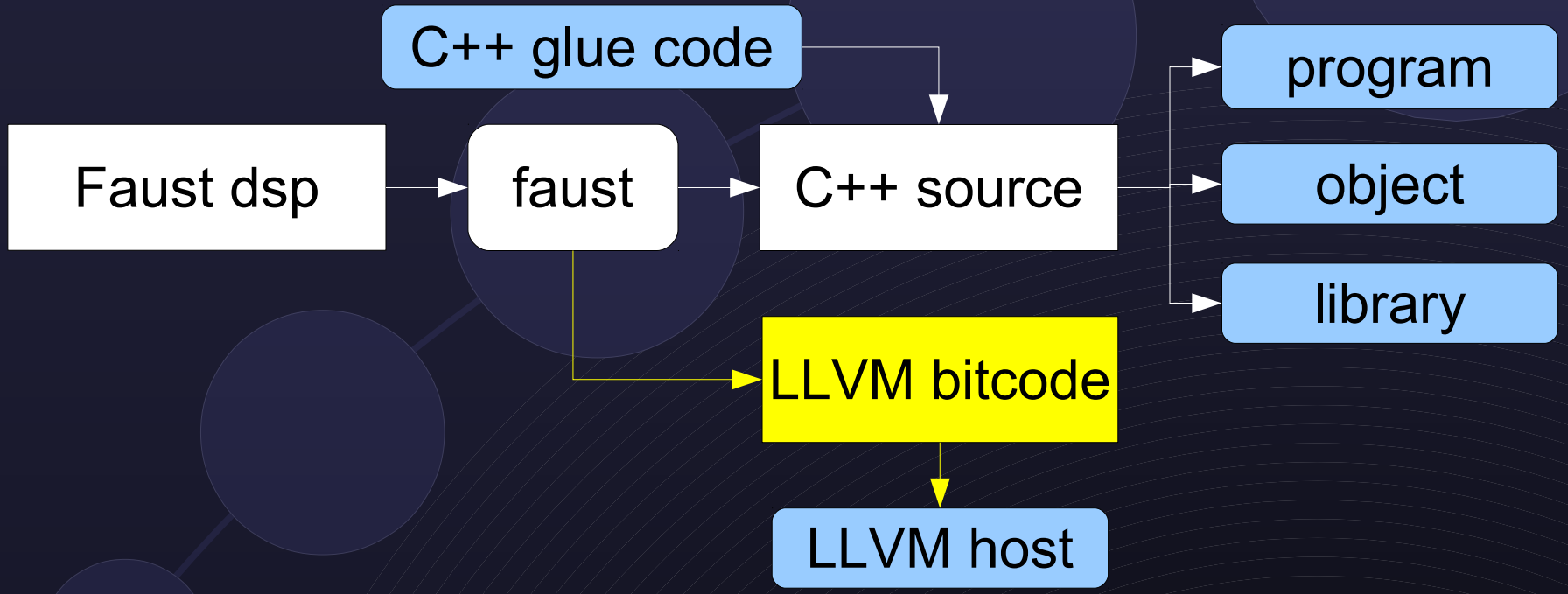
An LLVM bitcode interface between Pure and Faust

Albert Gräf
Department of Music Informatics
Johannes Gutenberg University Mainz

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

- **Faust** (“functional audio streams”): programs define the block diagrams of signal processors
  - turns functional descriptions into dsp code
  - supports many different host environments
- **Pure** (“pure universal rewriting engine”): programs are symbolic rewriting systems
  - “functional scripting language”, JIT-compiled
  - modern FP syntax + Lisp-like dynamic typing and metaprogramming capabilities
  - interfaces nicely to C, C++, Fortran, Octave, Pd, …
  - built-in vector/matrix data structure

- **LLVM** (“low-level virtual machine”): cross-platform compiler backend
  - JIT (just in time) and static compilation
  - fairly low-level code model, good for dsp
  - sophisticated optimizations, also at link time
  - used by llvm-gcc, clang, ghc, OpenCL, …



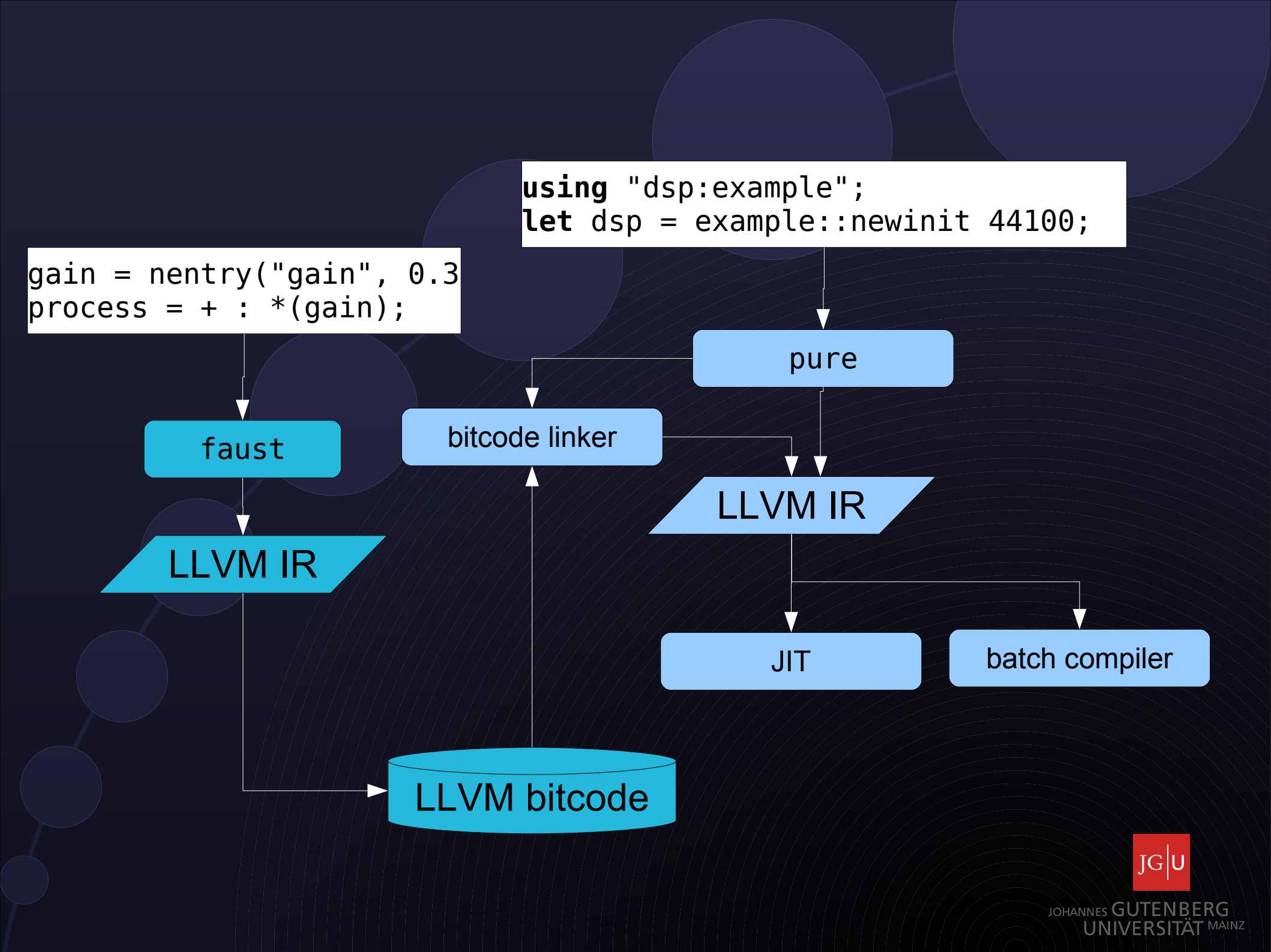
- **Faust LLVM backend** by Stéphane Letz (2010); see http://www.grame.fr/~letz/faust_llvm.html
- **Direct linkage** with LLVM bitcode
- **Dynamic loading** of Faust modules

The Pure-Faust Interface

- **Basic goal:** ability to run Faust dsps in Pure
- Somewhat like Snd-RT, but without restricting the host language
- Host only does “soft realtime”, but we still strive for low turnaround times to enable livecoding
- **Old interface:** Compile Faust module to a *shared library*, load in Pure via pure-faust module
  - clunky, needs C++ as intermediate language
  - high compilation times, not good for livecoding

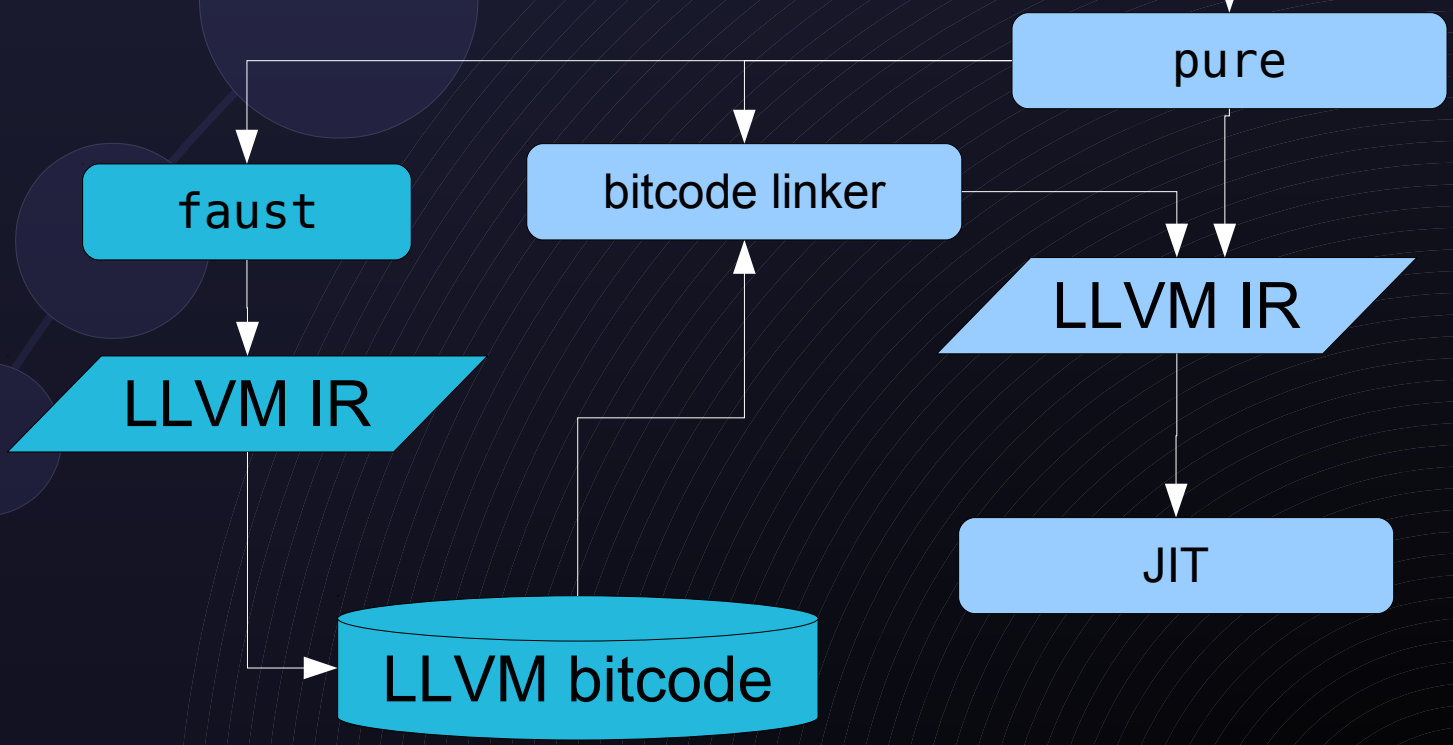
The Pure-Faust Interface

- **New interface:** Compile Faust module to *LLVM bitcode*, which can be loaded **directly** in Pure
  - possible to **inline** Faust code in Pure
  - faster turnaround, good (enough) for livecoding
- Benefits for the Faust programmer:
  - Use Pure as an interactive frontend to Faust
  - Use Pure to interface Faust to other systems

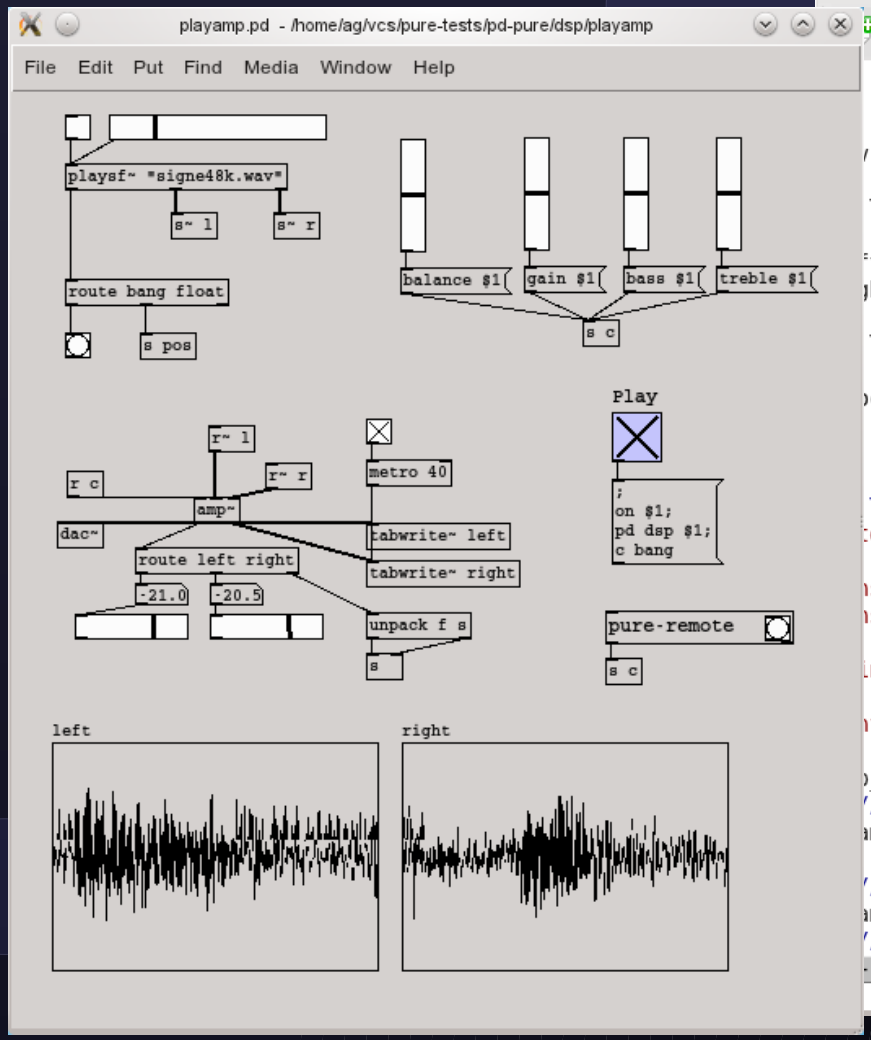

using "dsp:example";
let dsp = example::newinit 44100;
gain = nentry("gain", 0.3
process = + : *(gain);
pure
faust
bitcode linker
LLVM IR
LLVM IR
JIT
batch compiler
LLVM bitcode

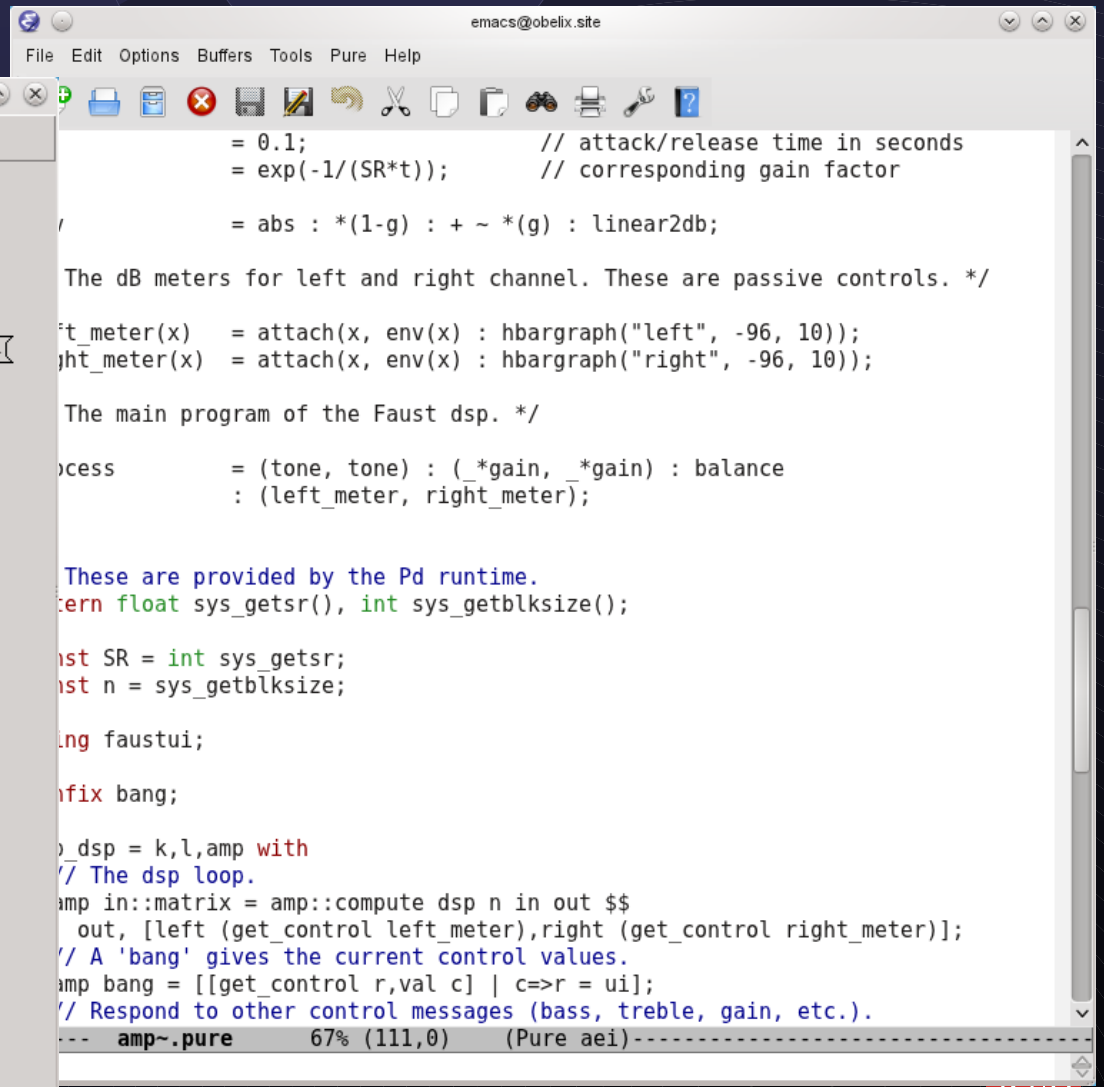
Inlining

```
%< -*- dsp:example -*-
gain = nentry("gain", 0.3, 0, 10, 0.01);
process = + : *(gain);
%>
let dsp = example::newinit 44100;
```



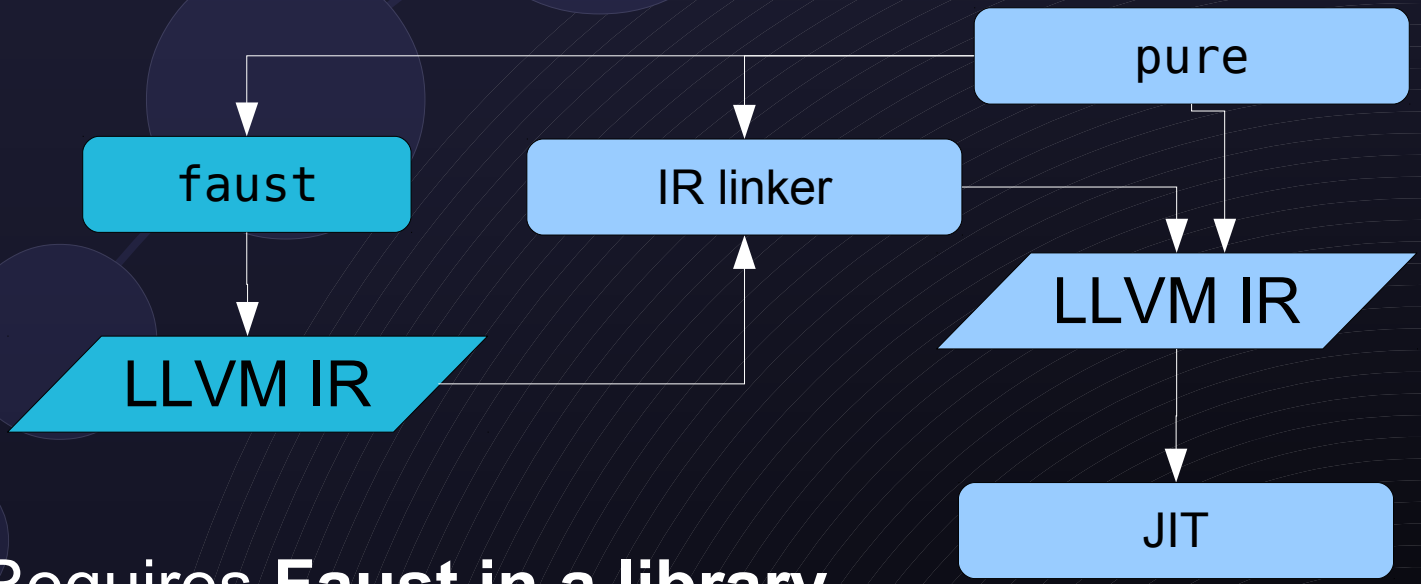
Examples


playamp.pd - /home/ag/vcs/pure-tests/pd-pure/dsp/playamp
File Edit Put Find Media Window Help
playsf~ "signe48k.wav"
s~ l
s~ r
route bang float
s pos
balance $1
gain $1
bass $1
treble $1
s c
Play
r~ l
r~ r
r c
amp~
dac~
metro 40
tabwrite~ left
tabwrite~ right
route left right
-21.0
-20.5
unpack f s
s
;
on $1;
pd dsp $1;
c bang
pure-remote
s c
left
right


emacs@obelix.site
File Edit Options Buffers Tools Pure Help
= 0.1; // attack/release time in seconds
= exp(-1/(SR*t)); // corresponding gain factor
= abs : *(1-g) : + ~ *(g) : linear2db;
The dB meters for left and right channel. These are passive controls. */
t_meter(x) = attach(x, env(x) : hbargraph("left", -96, 10));
ht_meter(x) = attach(x, env(x) : hbargraph("right", -96, 10));
The main program of the Faust dsp. */
cess = (tone, tone) : (_*gain, _*gain) : balance
: (left_meter, right_meter);
These are provided by the Pd runtime.
ern float sys_getsr(), int sys_getblksize();
st SR = int sys_getsr;
st n = sys_getblksize;
ng faustui;
fix bang;
_dsp = k,l,amp with
/ The dsp loop.
mp in::matrix = amp::compute dsp n in out $$
out, [left (get_control left_meter),right (get_control right_meter)];
/ A 'bang' gives the current control values.
mp bang = [[get_control r,val c] | c=>r = ui];
/ Respond to other control messages (bass, treble, gain, etc.).
--- amp~.pure 67% (111,0) (Pure aei)

Future Work

- Tighter integration via LLVM IR (skip bitcode files)



- Requires **Faust in a library**
- Faust as an **embedded sublanguage** in Pure (skip generation of Faust source)